**Lab 3: Clustering Using Python**

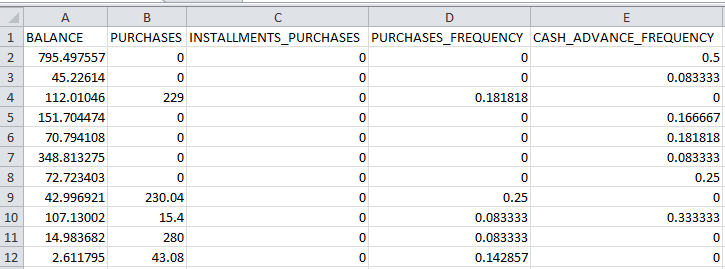
**What to submit:** a single word/pdf file with answers for the questions in **Part 5 (Try it yourself)**.

# Before you start

You’ll need two files to do this exercise: **Lab3.py** (the Python script file) and **CC GENERAL.csv** (the data file). All the files can be found on the Blackboard. (*Download all the files and save them to the folder where you keep your Python files.)*

# Part 1: Look at the Data File

1. Open the **CC GENERAL.csv** data file in Excel. If it warns you, that’s ok. Just click “Yes” and/or “OK.” You’ll see something like this:



This is the raw data for our analysis. The sample Dataset summarizes the usage behavior of about 8,600 active credit card holders during the last 6 months. The file is at a customer level with 5 behavioral variables.

In this type of input data file for a Cluster analysis, each row represents an observation, and each column describes a characteristic of the observation. We will use this dataset to create groups (clusters) of similar customers, based on these descriptor variables as dimensions. A customer of a cluster should be more similar to the other customers in its cluster than customers in any other cluster.

Following is the Data Dictionary for Credit Card dataset:

**BALANCE**: Balance amount left in their account to make purchases

**PURCHASES**: Amount of purchases made from account

**INSTALLMENTS\_PURCHASES**: Amount of purchase done in installment

**PURCHASES\_FREQUENCY**: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)

**CASH\_ADVANCE\_FREQUENCY**: How frequently the cash in advance being paid, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)

1. Close the **CC GENERAL.csv** file. If it asks you to save the file, choose “Don’t Save”.

# Part 2: Explore the Clustering.r Script

1. Open the **Lab3.py** file in PyCharm. This contains the Python script that performs the clustering analysis.
2. Look at lines 3 through 11. These install (when needed) and load the **cluster** and **other** packages. These perform the clustering analysis and visualization.
3. Look at line 14 through 15. Set working directory. Change it to your working folder.

os.chdir(r"C:\Users\Shuting Wang\Dropbox\Teaching\Baruch\Teaching2023 spring\CIS9660\Labs\Clustering")

1. Look at Line 19. Import your data for the analysis.

mydata = pd.read\_csv('CC GENERAL.csv')

1. Remove outliers. Look at line 21~27.

We are defining outliers as values that are more than three standard deviations away from the mean.

#Remove outliers  
def remove\_outliers(df, z\_thresh):  
 z\_scores = np.abs((df - df.mean()) / df.std())  
 df = df[(z\_scores < z\_thresh).all(axis=1)]  
 return df  
  
x = remove\_outliers(mydata, 3.0)

1. Normalize the data. Look at line 30~33.

# Normalize the data using StandardScaler  
scaler = StandardScaler()  
normalized\_data = scaler.fit\_transform(x)  
x = pd.DataFrame(normalized\_data, columns=mydata.columns)

1. Look at line 37~41. This command creates a **K-means clustering model**:

kmean=KMeans(n\_clusters=5, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001, random\_state=0)  
kmean.fit(x)  
kmean.cluster\_centers\_  
labels=kmean.labels\_

The basic format of this command is:

KMeans(n\_clusters=5, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001, random\_state=0)

The **KMeans()** is used to run the K-means clustering algorithm

1. n\_clusters: This parameter specifies the number of clusters that the algorithm should try to identify in the data. It is a required parameter.
2. init: This parameter specifies the method for initializing the initial centroids of the clusters. The possible values are 'k-means++', 'random', or a user-supplied array of centroids.
3. n\_init: This parameter specifies the number of times the algorithm should be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.
4. max\_iter: This parameter specifies the maximum number of iterations that the algorithm should run for each initialization of the centroids.
5. tol: This parameter specifies the tolerance for convergence. If the difference between the previous and current centroid positions is less than tol, the algorithm stops.
6. random\_state: This parameter specifies the seed used by the random number generator.
7. Look at line 46. This command calculates the within cluster sse:

# Calculate the withinness-cluster SSE (sum of squared errors)  
within\_cluster\_sse = kmean.inertia\_

1. Look at line 49. This command calculates the Silhouette Score:

silhouette\_avg = silhouette\_score(x, labels)

# Part 3: Reading Output

1. Now open the file **output.txt**. (*It will also be in the folder with your Lab3.py script.*)
2. **Reading Summary Statistics.**

The first thing you’ll see are the **summary statistics for each variable**:

BALANCE PURCHASES \

mean std count mean std count

labels

0 2.367027 0.761954 725 -0.176860 0.731174 725

1 -0.445963 0.503797 2702 0.082897 0.606131 2702

2 0.194673 1.029381 849 2.159606 1.256459 849

3 -0.319040 0.548175 2632 -0.483428 0.335576 2632

4 0.136109 0.571218 1200 -0.547406 0.313731 1200

INSTALLMENTS\_PURCHASES PURCHASES\_FREQUENCY \

mean std count mean std

labels

0 -0.301322 0.585422 725 -0.330073 0.946015

1 0.170401 0.599951 2702 0.903531 0.440180

2 2.162462 1.321650 849 1.105362 0.340969

3 -0.539140 0.190518 2632 -0.796555 0.427203

4 -0.549065 0.241091 1200 -0.869964 0.617788

CASH\_ADVANCE\_FREQUENCY

count mean std count

labels

0 725 1.162980 1.107609 725

1 2702 -0.533906 0.425712 2702

2 849 -0.341026 0.724137 849

3 2632 -0.375766 0.390944 2632

4 1200 1.565001 0.776672 1200

Now let’s look at the normalized values for cluster (group) 0:

As seen, the averages of Purchases, Installments\_purchases, and Purchases\_frequency for group 0 are negative, thus are below the population average (i.e., 0). In other words, the customers in cluster 0 make less purchases, less installments purchases, and purchase less frequently than the overall population.

1. **Within-Cluster SSE (Cohesion) and Silhouette Score**

We want to better understand the “quality” of the clusters. Let’s look at the within-cluster sum of squares error (i.e. within-cluster SSE). In Python, it is called “**inertia**.” The within-cluster SSE measures **cohesion** – how similar the observations within a cluster are to each other.

We also use silhouette score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette\_score.html) to indicate the quality of the resulting clusters. A silhouette value is a combination of two scores: cohesion and separation.The Silhouette score is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette score for a sample is (b - a) / max(a, b). To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of.

The best value is 1 and the worst value is -1. The silhouette score of 1 means that the clusters are very dense and nicely separated. The score of 0 means that clusters are overlapping. The score of less than 0 means that data belonging to clusters may be wrong/incorrect.

The following are the lines which contain that statistic:  
  
Within-cluster SSE: 14340.876575041657

Silhouette Score: 0.31791104557295874

The within-cluster-sse of this clustering results is 14340.87 and the Silhouette Score is 0.32. We can use these to compare this set of clusters to another set of clusters we will create later using the same data.

# Part 4: Comparing Two Sets of Clustering Results

Now we’re going to create another set of clusters (10 clusters instead of 5) and examine the withinss and betweenss to understand the tradeoff between the number of clusters, cohesion, and separation.

1. Open the Lab3.py file in PyCharm.
2. Look at line 38:  
     
   kmean=KMeans(n\_clusters=5, init='k-means++', n\_init=10, max\_iter=300, tol=0.0001, random\_state=0)  
     
   Change this value from 5 to 10.
3. Look at line 54:

writer = pd.ExcelWriter('clustering\_results.xlsx', engine='xlsxwriter')

Change “'clustering\_results.xlsx” to “'clustering\_results10.xlsx”.

1. Look at line 61:

file = open('output.txt','wt')

Change “output.txt” to “output10.txt”.

1. Re-run the script.
2. Open output10.txt.
3. You’ll notice now, in the cluster means section there are 10 clusters:

BALANCE PURCHASES \

mean std count mean std count

labels

0 0.356341 1.155812 302 2.614096 1.130579 302

1 -0.541957 0.388091 1823 -0.117119 0.419565 1823

2 0.002132 0.591839 1528 -0.641413 0.167427 1528

3 -0.408075 0.467353 655 0.783623 0.503579 655

4 -0.538315 0.392320 1673 -0.330244 0.384704 1673

5 0.333301 0.728247 387 0.166116 0.592754 387

6 2.068296 0.798682 323 0.552305 0.809809 323

7 0.182391 0.573403 666 -0.599923 0.249249 666

8 2.525081 0.708395 443 -0.571926 0.246164 443

9 0.025573 0.820124 308 2.771383 1.097938 308

INSTALLMENTS\_PURCHASES PURCHASES\_FREQUENCY \

mean std count mean std

labels

0 3.405320 0.900967 302 1.153876 0.300416

1 0.018875 0.416372 1823 0.974313 0.365358

2 -0.617892 0.065617 1528 -1.115628 0.232677

3 1.616937 0.633006 655 1.037423 0.396652

4 -0.445279 0.263857 1673 -0.478950 0.407190

5 0.138325 0.659319 387 0.778130 0.470076

6 0.336925 0.839619 323 0.723707 0.605050

7 -0.604893 0.100654 666 -1.047919 0.309157

8 -0.574110 0.188791 443 -0.941868 0.512682

9 0.201339 0.741819 308 0.915830 0.518703

CASH\_ADVANCE\_FREQUENCY

count mean std count

labels

0 302 -0.345634 0.740721 302

1 1823 -0.617269 0.247355 1823

2 1528 0.178660 0.427173 1528

3 655 -0.571488 0.350991 655

4 1673 -0.645217 0.195334 1673

5 387 1.552510 0.801923 387

6 323 0.085927 0.800596 323

7 666 1.937409 0.668372 666

8 443 1.437695 1.022483 443

9 308 -0.471902 0.527546 308

Cluster 8 has the highest balance, while cluster 9 has the highest average purchases. Because these values are **normalized**, you aren’t looking at the actual values (i.e., the number of purchases a customer made). But it does let you compare clusters to each other.

Most importantly, we can compare the withinss and Silhouette statistics for this new set of clusters to our previous configuration of 5 clusters:

Within-cluster SSE: 9344.206590624122

Silhouette Score: 0.29041241690651165

We can see that the within-cluster sse is 9344.22 and the Silhouette Score is 0.29

1. **Compare the 10 cluster solution to our 5 cluster solution**, where the withinss is 14340.88. The total withinss error is clearly lower for our 10-cluster solution; **the clusters in the 10 cluster solution have higher cohesion than the 5 cluster solution**. This makes sense – if we put our observations into more clusters, we’d expect those clusters to (1) be smaller and (2) more similar to each other.

However, we can see that the separation is lower (i.e., worse) in our 10-cluster solution. For the 10-cluster solution, the Silhouette Score is 0.29; for the 5 cluster solution, the Silhouette Score is 0.32. **This means the clusters in the 10 cluster solution are worse than the 5 cluster solutions.**

The relationship between the number of clusters (k) and the Silhouette Score is not always straightforward. In general, as the number of clusters increases, the Silhouette Score may increase or decrease depending on the quality of the clustering.

***Question: How many clusters should we choose?***

# Part 5: Try it yourself

Use the Lab3.py script and the same CC GENERAL.csv dataset to create a set of 7 clusters:

1. What is the size of the largest cluster?

The largest cluster #3 has 2443 observations.

1. Compare the characteristics of cluster 3 to the population as a whole?

Cluster 3 has a lower balance, lower purchases, lower installments-purchases, lower purchase frequency, and lower cash-advance\_frequency than the overall population average.

1. What is the withinss error for those 7 clusters?

11910.01419105354

1. Is the cohesion generally higher or lower than the 5-cluster solution?

Cohesion is generally higher than in the 5-cluster solution.

1. What is the Silhouette score for those 7 clusters? What does it mean, compared to the Silhouette score for those 10 clusters?

Silhouette Score: 0.30560207658565247, which is larger than 0.29 of 10 clusters. This means that clusters are better when k-7.

1. Use your own words to summarize how the number of clusters can affect cohesion and silhouette score.

Increasing the number of clusters always results in (1) higher cohesion. However, the relationship between the number of clusters (k) and the Silhouette Score is not always straightforward. In general, as the number of clusters Increases, the Silhouette Score may increase or decrease depending on the quality of the clustering.